

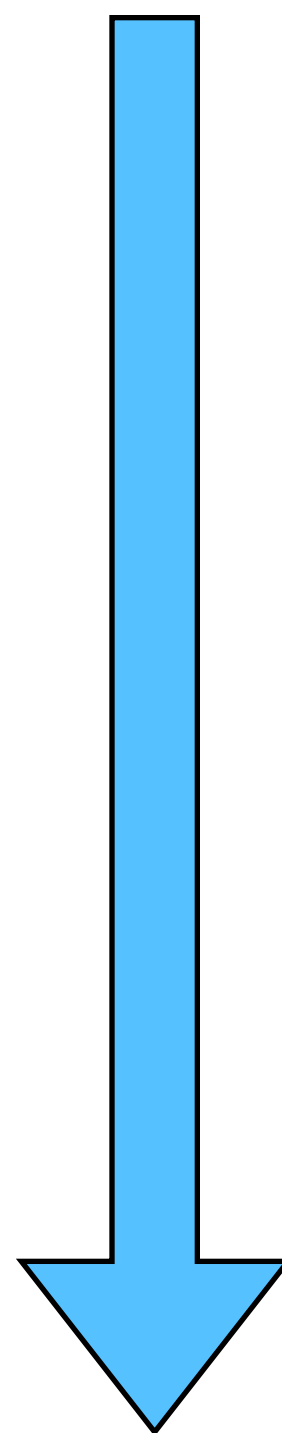
SVFS Limitations

Outline

- Performance limitations of SVFS
- Strategies to improve FS performance
- Other limitations of SVFS

Review: Directory Lookups

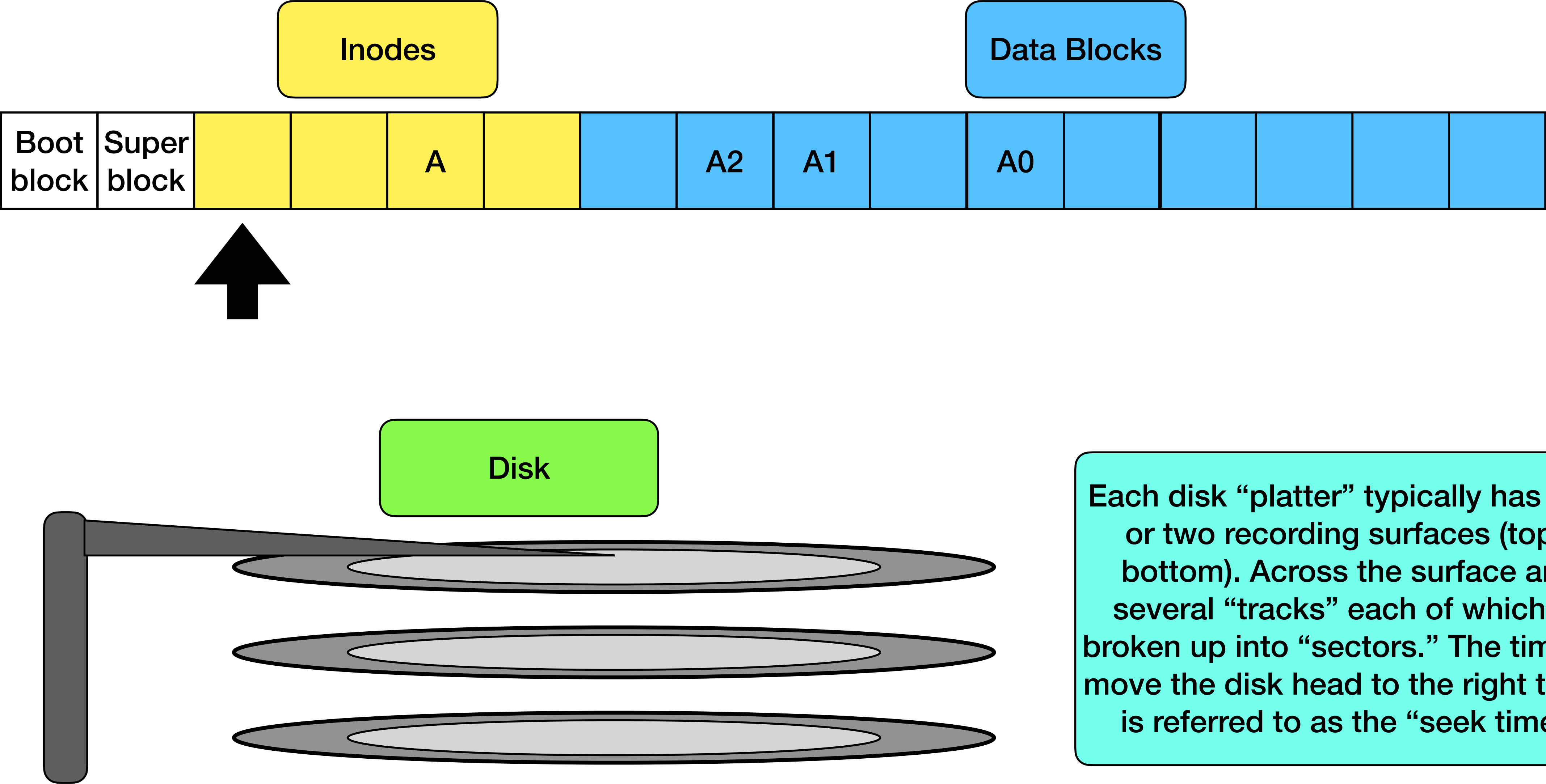
Linear scan
through directory
data is slow to
find a file!



Component Name	Inode Number
.	1
..	1
home	117
usr	4
bin	47
etc	36

For file systems with large
directories, we can organize our
directories as a B-tree → use
hashing to lookup a child path!

SVFS Performance



Chat with your neighbor(s)!

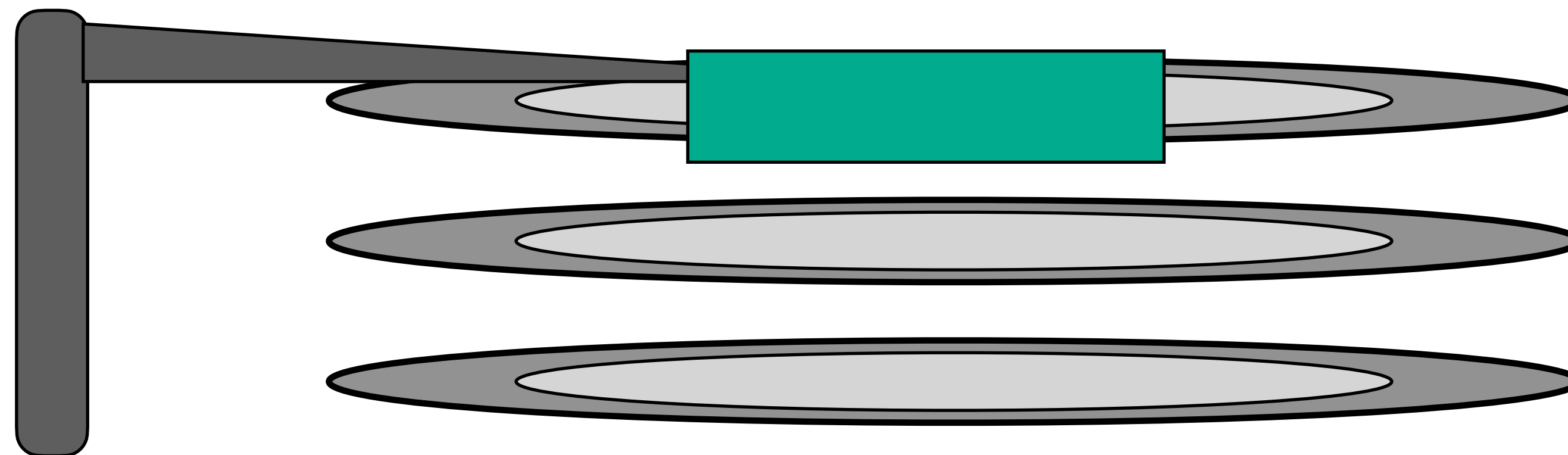
The SVFS suffers long seek times (e.g., the time to move the disk head to the sector holding data). Describe the best case scenario to achieve optimal disk performance. Think about the location of the data on the disk that we are reading from/writing to.

SVFS Performance

- The SVFS suffers significant performance degradation due to long seek times waiting for the disk head to make its way around the platters, and transfers of file data are typically very slow
- Data blocks are allocated on disk from *free lists* in the SVFS, which does not consider the placement of data blocks relative to one another ➡ data is generally placed on disk wherever there is available space
- Consider the procedure to look up a large file from an indirect block: read the inode to find the indirect block, read the indirect block to find the data, read the inode again to read the doubly indirect block, etc...
- On average, SVFS transfers data at about 0.16% of HDDs maximum transfer speed!

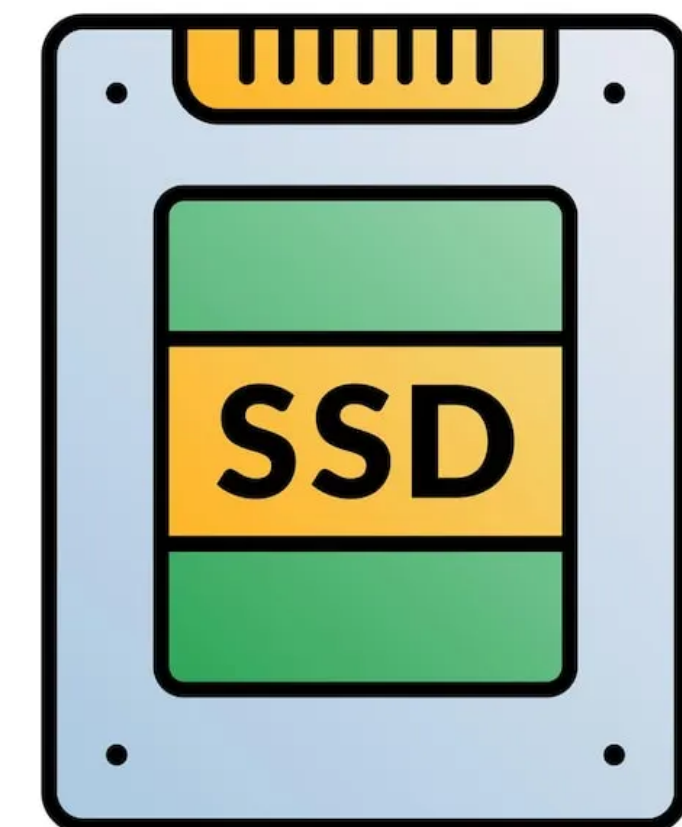
Strategy 1: Improving Disk Performance (Hardware)

Pre-Fetch Buffer



Option 1: improve
device by caching

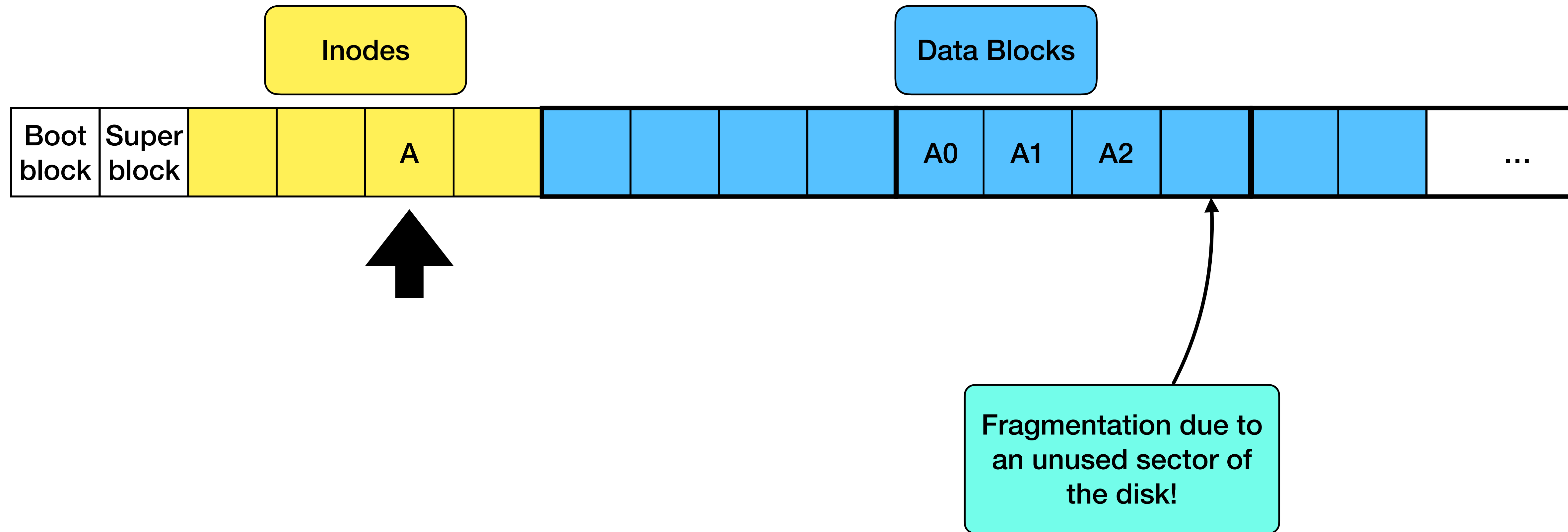
Option 2: use
devices that are
better suited for task



Random access
storage device

Sequential operations
remain much more efficient

Strategy 2a: Larger Data Blocks



Strategy 2a: Larger Data Blocks in FFS/ext2

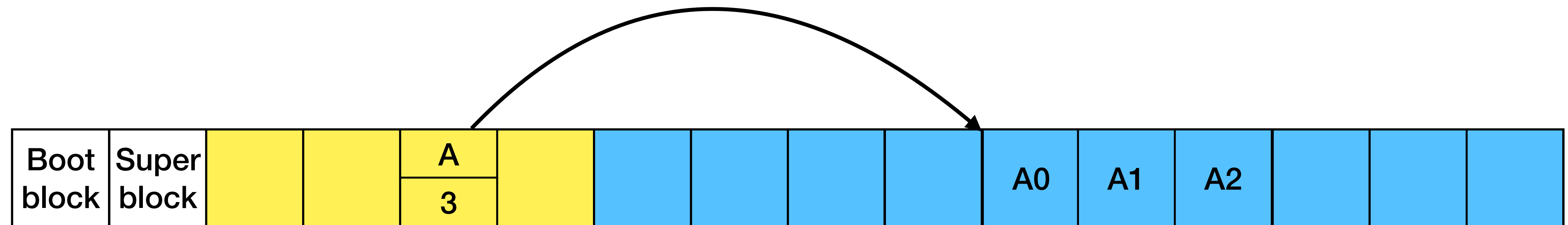
- Each “data block” referred to be an inode is composed of eight 512-byte sectors on the disk
- To reduce fragmentation on disk, the last data block of a file can be expressed as a “partial data block” in which the file is composed of some number of n contiguous sectors within a block
- Partial data blocks need to be tracked in a separate free list, and the FFS switches allocation strategies to revert back to SVFS mapping for the remaining free sectors

Chat with your neighbor(s)!

Suppose we wanted to implement a data block allocation scheme like that of FFS/ext2. What would need to change about the inode struct to implement both data block allocation strategies?

The inode would need to have a mechanism to distinguish whether the data block is a “whole block” or “sector”

Strategy 2b: Clustering and Extents



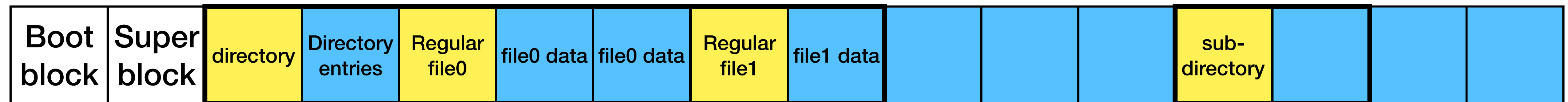
Length of file

Files may be allocated as one or more “extents” or clusters of blocks that are referred to by the inode

If we want to append to a file and add a data block, then we need to add a new “extent” to the inode

This strategy is used in NTFS (Windows OS)!

Strategy 3: Reduced Seek Latency



Organize disk in terms of directory groups to reduce seek latency within a directory!

Subdirectories get their own groups of data blocks/sectors!

Strategy 3: Reduced Seek Latency in FFS/ext2

- In the System V File System, the disk has a region specifically assigned to maintaining inode blocks and data blocks, but these sectors are not physically proximate to one another
- In FFS and ext2, the whole disk is considered to be a sequence of sectors/data blocks that may be allocated as either an inode or data
- To implement this, groups of files associated with a directory are colocated with one another to make for faster directory reads ➡ each directory/subdirectory gets its own chunk of disk to operate on
- This means that the operating system needs to track an additional free list!

Chat with your neighbor(s)!

Consider the construction of the FFS/ext2 disk layout in which sectors are organized into “file groups” according to their directory. What is the maximum number of inodes in such a directory?

SVFS Hard Coded Constraints

- In the System V File System, space is allocated for the all of the inodes in advance, and inodes have a well defined format
- As a result of needing to know the maximum number of inodes a priori, the maximum number of files that can be created by the system must also be know in advance...
- As a result of predefined inode format (e.g., direct, singly-indirect, doubly-indirect, etc...), the maximum file size has to be known in advance...
- Each of these features are a limitation of the usability of this file system!

Other Desirable File System Features

- Journaling in memory (explicit support for writing to an mobj) and crash consistency
- File update atomicity
- Mechanisms to check the status of a file system and correct errors

From Wednesday: Freeing Vnodes (demo)

```
[stum2025@itbdcv-lnx04p demo11]$ pwd
/home/stum2025/cs134/demos/demo11
[stum2025@itbdcv-lnx04p demo11]$ mkdir temp
[stum2025@itbdcv-lnx04p demo11]$ touch temp/test1; touch temp/test2; touch temp/test3
[stum2025@itbdcv-lnx04p demo11]$ rm -rf temp
[stum2025@itbdcv-lnx04p demo11]$ ls
[stum2025@itbdcv-lnx04p demo11]$ stat /home/stum2025/cs134/demos/demo11/temp
stat: cannot statx '/home/stum2025/cs134/demos/demo11/temp': No such file or directory
[stum2025@itbdcv-lnx04p demo11]$ tat .
-bash: tat: command not found
[stum2025@itbdcv-lnx04p demo11]$ stat .
  File: .
  Size: 6          Blocks: 0          IO Block: 4096   directory
Device: fd02h/64770d  Inode: 889040658  Links: 2
Access: (0755/drwxr-xr-x)  Uid: (989988336/stum2025)   Gid: (989800513/domain users)
Context: unconfined_u:object_r:user_home_t:s0
Access: 2026-10-07 13:54:06.041410641 -0700
Modify: 2026-10-07 13:53:59.099705445 -0700
Change: 2026-10-07 13:53:59.099705445 -0700
 Birth: 2026-10-07 11:16:13.710867842 -0700
[stum2025@itbdcv-lnx04p demo11]$ █
```

```
[stum2025@itbdcv-lnx04p demo11]$ pwd
/home/stum2025/cs134/demos/demo11
[stum2025@itbdcv-lnx04p demo11]$ cd temp/
[stum2025@itbdcv-lnx04p temp]$ ls
test1  test2  test3
[stum2025@itbdcv-lnx04p temp]$ ls
[stum2025@itbdcv-lnx04p temp]$ pwd
/home/stum2025/cs134/demos/demo11/temp
[stum2025@itbdcv-lnx04p temp]$ stat /home/stum2025/cs134/demos/demo11/temp
stat: cannot statx '/home/stum2025/cs134/demos/demo11/temp': No such file or directory
[stum2025@itbdcv-lnx04p temp]$ stat .
  File: .
  Size: 6          Blocks: 0          IO Block: 4096   directory
Device: fd02h/64770d  Inode: 687587473  Links: 0
Access: (0755/drwxr-xr-x)  Uid: (989988336/stum2025)   Gid: (989800513/domain users)
Context: unconfined_u:object_r:user_home_t:s0
Access: 2026-10-07 13:53:43.869158195 -0700
Modify: 2026-10-07 13:53:59.099705445 -0700
Change: 2026-10-07 13:53:59.099705445 -0700
 Birth: 2026-10-07 13:53:30.441794128 -0700
[stum2025@itbdcv-lnx04p temp]$
```

Chat with your neighbor(s)!

To create a `vnode` in the file system, we allocate the space in kernel memory and insert the created structure into the `vnode_list` in the file system. How do we know when a `vnode` can be destroyed? Think about what happens if one or more processes each have a file descriptor pointing to that file.

If a process refers to a `vnode`, we cannot destroy it

To track how many references to a `vnode` there are, the `vnode` itself needs to track a count of “references” pointing to it

When creating an entry in the process file descriptor table, we need to also increment the `vnode` reference count!

Summary

- The System V File System performs very slowly! In general, these slow features are a function of the directory layout and the organization of data blocks on disk
- To improve disk layout, we can organize disk sectors into contiguous blocks so that files may be read as singular consecutive reads
- To further improve disk layout, we can “unrestrict” the number of data sectors that an inode refers to by using extents (as is done in NTFS)
- If we use a directory-aware allocation of data blocks, then we can further reduce seek time within a directory